AN INTRODUCTION TO

AUDIO PLUGIN DEVELOPMENT AND REALTIME AUDIO PROGRAMMING

Ryan Gildea

March 15, 2019

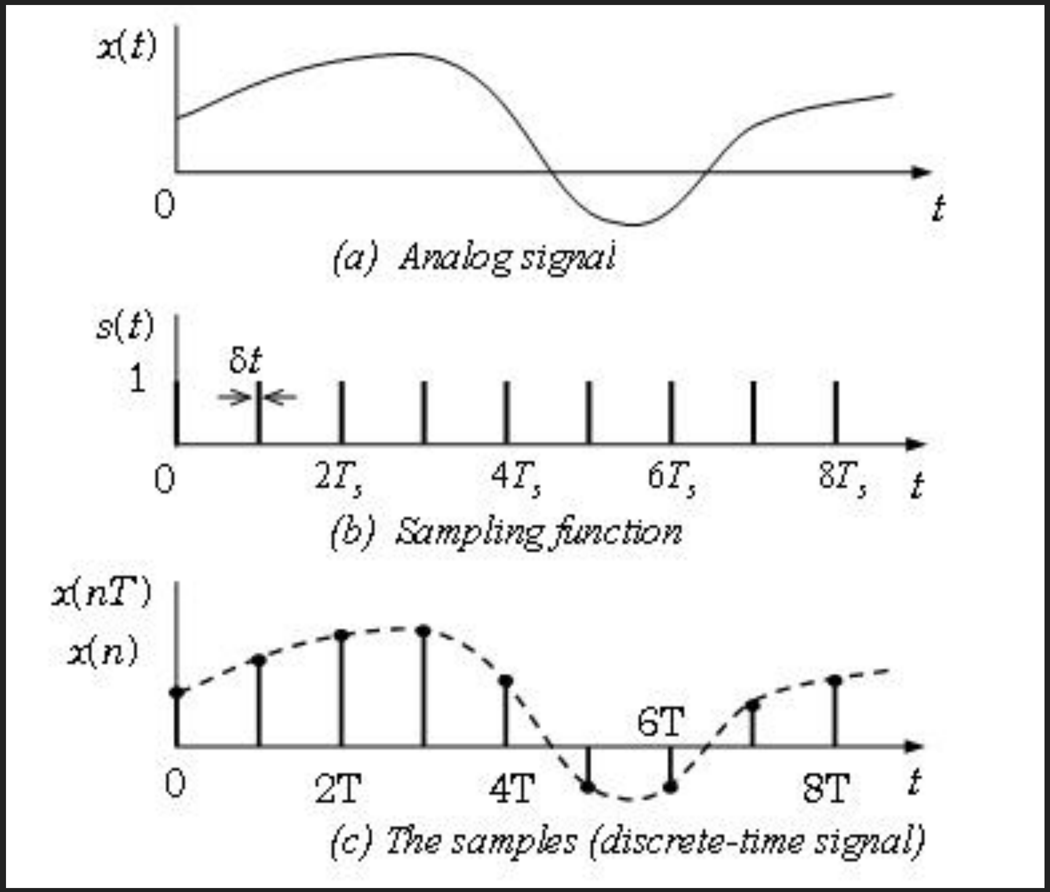
WHAT IS AN AUDIO PLUGIN?

- Plugins are software written to generate or process audio signals in real-time inside of a Digital Audio Workstation.
- Effect Plugins allow you to alter a signal to produce a different sounding result.
- E.g., Equalization, Dynamics Processing, Modulation, Reverb, Delay, Pitch Correction, Acoustic Modeling, etc.

DIGITAL AUDIO SOFTWARE BASICS

- Host - Digital Audio Workstation software, allowing the user to record, edit, sequence, manipulate and combine multiple digital audio files.
- Plugin - Software extension to process a digital audio signal in realtime.
- Plugin Parameter - A value defined and used by a plugin that a host may read and/or write.
- Parameter Automation - Storing and replaying changes to parameter values over time.
- Control Surfaces - External hardware to modify parameters and input MIDI notes.

HOW IS AN AUDIO SIGNAL REPRESENTED DIGITALLY?



source: https://signalprocessingsampling.weebly.com/

- Continuous vs. Discrete Signals
  - Analog signals are continuous; within the waveform there is an infinite amount of information
  - digital signals are signals with discrete values (samples) at specific time intervals
  - Sample Rate - # of samples collected of a continuous signal per second (Hz)

HOW ARE SIGNALS PROCESSED?

512	512	512	512	512	512	512

- Block Size - fixed number of samples that are processed at once by the Audio Host & plugins
- A buffer (array) of samples of the configured block size is processed together.
- Control signals and other parameters are measured at a point in time, and the audio is processed using those values, and the resulting sequence of samples is written into the buffer.
- Then host processes the next chunk of samples determined by the block size.

TENETS OF REALTIME AUDIO PROGRAMMING

- Ross Bencina's rules of thumb for real-time audio callback programming:
- Don't allocate or de-allocate memory.
- Don't lock a mutex.
- Don't read or write to the file system or otherwise perform i/o.
- Don't call OS functions that may block waiting for something.
- Don't execute any code that has unpredictable or poor worst-case timing behavior.
- Don't call any code that does or may do any of the above.
- Don't call any code that you don't trust to follow these rules.
- Source: http://www.rossbencina.com/code/real-time-audio-programming-101-time-waits-for-nothing

DESIGN PATTERNS FOR AUDIO APPLICATIONS

- The Observer Pattern
  - Allow an object (producer) to notify other objects (observers) of state changes in a scalable and efficient manner.
  - Observer pattern is common in many different areas of software development, especially user interfaces, e.g. mobile application UI.
  - Observers register with Producer
  - Producer maintains list of Observers, and notifies them of state changes.
    - E.g., calling a callback function or posting a message to a message queue to be read and processed on another thread.

DESIGNING THE DELAY ALGORITHM

- Tape Loop Analogy
  - Tape Loop
    - Circular Buffer
  - Read Head - Pointer to where in the buffer we are reading out of the circular buffer.
  - Write Head - Pointer to where in the buffer we are writing into the circular buffer.
- The distance (in samples) between the read head and write head determine delay time

source: http://www.bruceduffie.com/ussachevsky.html

BASIC IMPLEMENTATION

- Parameters
  - Delay Time
    - Amount of time in seconds between original signal and delayed (wet) signal.
  - Wet/Dry Ratio
    - Ratio of volume between Wet (delayed) signal and Dry (original) signal.
  - Feedback (Ratio)
    - Amount of wet signal to feed back into the input of the delay function, creating repeats.
- In the process block:
  - read each sample and copy it to the correct position in the circular buffer,
  - read from the proper position in the circular buffer
  - Generate feedback

IMPROVEMENTS

- Interpolation
  - Sometimes the sample offset (delay time in seconds * sample rate) is not an integer.
  - What value do we use then?
- Linear interpolation = line between two values over an interval.
  - we can use linear interpolation to estimate a suitable value to use.

```
float KadenzeDelayAudioProcessor::lin_interp(float inSampleX, float inSampleY, float inFloatPhase)
{
    return (1 - inFloatPhase) * inSampleX  + inFloatPhase * inSampleY;
}
```

IMPROVEMENTS

- Parameter Smoothing
  - Drastic changes in parameter values can cause audio discontinuities, which sound harsh and unpleasant.
  - When accepting user input, smoothing is sometimes necessary to avoid undesirable audio discontinuities caused by values jumping very quickly.
  - Instead of jumping immediately to the value specified by the UI parameter, we can adjust the value gradually (per sample loop iteration) using this formula:

```
for(int i = 0; i < buffer.getNumSamples(); i++) {
    mTimeSmoothed = mTimeSmoothed - 0.0001*(mTimeSmoothed - *mTimeParameter);
    mDelayTimeInSamples = getSampleRate() * mTimeSmoothed;
```

QUESTIONS?

ACKNOWLEDGMENTS

- Based on course material created by Jacob Penn, Output & kadenze.com
- https://www.kadenze.com/courses/intro-to-audio-plugin-development
- Thank you!